UNIVERSIDADE FEDERAL DE MINAS GERAIS

PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARES 1

Documentação – Trabalho prático
“R-Type”

Mariana Assis Ramos

Belo Horizonte
Fevereiro/2022

- **INTRODUÇÃO**

Esse documento tem como objetivo explicar a implementação e o funcionamento do jogo R-type referente ao Trabalho Prático.

- **DESCRIÇÃO DO JOGO**

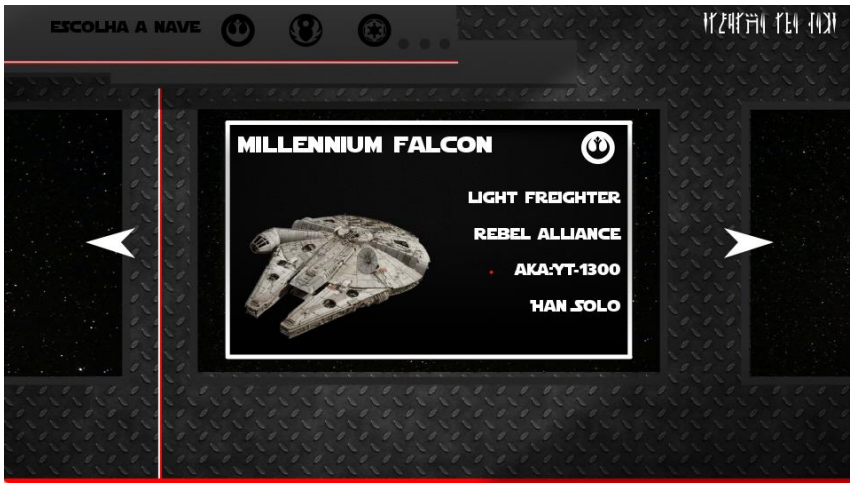
  - **Menu**

O jogo, que tem como inspiração a saga de filmes StarWars, começa fazendo uma alusão a abertura dos filmes. Em seguida abre o menu inicial, no qual o jogador altera com as setas ou com as teclas W e S qual dos botões ele deseja selecionar, clicando ESPAÇO para confirmar a decisão. Os botões tem como opção “Começar”, "Como jogar" e "Sair do jogo".

O botão “Começar” direciona o jogador para um segundo menu em que ele precisa selecionar com qual nave vai jogar e a dificuldade do jogo. O botão "Como jogar" leva para uma tela contendo informações de como jogar e o "Sair do jogo" fecha o programa.

Ao apertar ESC o programa também é fechado.

Tela inicial

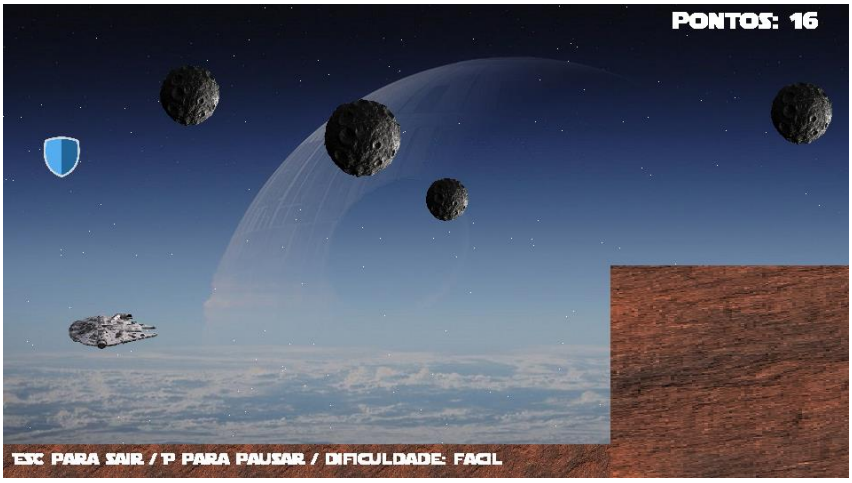


Escolha de nave e dificuldade

- **Jogo**

Após a escolha da nave e da dificuldade o jogador pode começar o jogo. Caso tenha escolhido a dificuldade fácil, existirão poderes ao longo do percurso, já no médio eles não existem.

Durante o jogo o objetivo do jogador é eliminar os cometas, por meio dos tiros da nave, aumentando a pontuação, e desviando dos blocos e dos cometas que ele não eliminar. Os tiros são controlados pelo ESPAÇO, existindo duas variações: tiro simples (só apertar a tecla) e tiro avançado (apertar e segurar). A nave é controlada com as teclas A W, S e D. Os poderes, que aparecem aleatoriamente, servem para dar imunidade contra os cometas. É possível pausar o jogo ao pressionar a tecla P.



Tela de jogo

- **Final do jogo**

Caso o jogador seja eliminado (colidir com bloco ou com cometas) o jogador é redirecionado a tela final que exibe sua pontuação e o recorde.

- **CONTROLES**

usa somente teclado

- **Durante o jogo**

A, S, W, D: move nave.
P: pausar o jogo.
ESPAÇO: atirar.
ESC: sair do jogo.

- **Nos menus**

A, S, W, D: alternar opções de menu.
SETAS: alternar opções de menu.
ESPAÇO: selecionar opções do menu;
ENTER: iniciar e recomeçar jogo.
ESC: sair do jogo.

- **DESCRIÇÃO DO CÓDIGO**
  Código está modularizado

- **Resumo**

O código do arquivo principal consiste em um loop do jogo que é subdividido em diferentes eventos. No caso do evento de “Time” é subdividido novamente em diversos "IF", referentes aos diferentes momentos (o jogo em si, o menu, o menu final...), esses “IF”s contam com funções responsáveis para o andamento do jogo. Os outros eventos servem para controlar a movimentação do menu e da nave durante o jogo de acordo com as teclas pressionadas. As funções referentes ao andamento do jogo estão no arquivo TP_funcoes.c.

- **TP_rtype.c**

Linha 1 a 12: Include bibliotecas usados no jogo.
Linha 14 a 27: Declaração de constantes.

Linha 32: INÍCIO MAIN Linha 38 a 69: declaração objetos Allegro

Linha 72 a 217: rotina de inicialização Allegro(Instalação add-nos Allegro; Declaração outros objetos Allegro;)

Linha 220 a 234: Criação fila de eventos(event_queue).

Linha 238 a 291: Declaração de objetos e variáveis do jogo.

Linha 301: INÍCIO WHILE; loop do jogo

Linha 303 a 304: EVENT_QUEUE.

Linha 310: event type TIMER.

Linha 314 a 327: GAMEPLAY == -1; refere ao modo de reset das funções do jogo

Linha 328 a 415: GAMEPLAY == 0; modo do jogo em si; conta com todas as funções que faz o jogo seguir

Linha 417 a 447: GAMEPLAY == 1; modo referente ao menu inicial com “botões” para submenus

Linha 448 a 487: GAMEPLAY == 2; modo referente ao menu final, com a highscore, score e opção de reset

Linha 488 a 516: GAMEPLAY == 3; modo com o submenu de ajudas, apresenta as instruções de como jogar

Linha 517 a 576: GAMEPLAY == 4; modo com submenu de escolher entre as 3 naves para começar o jogo

Linha 577 a 614: GAMEPLAY == 5; último submenu antes do jogo, onde o jogador escolhe a dificuldade, isto é, se que Power Up ou não

Linha 615 a 625: GAMEPLAY == 10; primeira tela do jogo, faz os instantes da introdução de Star Wars

Linha 631: event type DISPLAYCLOSE (acaba o jogo caso feche a tela)

Linha 636 a 937: event type KEY DOWN; checa quais teclas foram pressionadas pelo jogador e suas devidas funções;

Linha 938 a 973: event type KEY UP; checa quais teclas deixaram de ser pressionadas pelo jogador e suas devidas funções;

Linha 975 a 922: event type KEY CHAR; checa quais teclas foram pressionadas e seguradas pelo jogador e suas devidas funções;

Linha 995: FIM WHILE; loop do jogo

Linha 998 a 1037: rotinas de fim de jogo

Linha 1040: FIM MAIN

- **TP_funcoes.c**

Linha 1 a 10: Include bibliotecas usados no jogo.

Linha 26 a 627: todas as funções usadas no jogo:

Básico das funções:

→ Void init...: são funções que inicializam objetos do jogo, de acordo com qual struct eles são

→ Void desenha....: são funções que desenham os objetos do jogo, sejam eles bitmap ou formas geométricas.

- → Void atualiza....: são funções que normalmente atualizam a posição dos objetos durante o jogo, podendo ter outras utilidades dependendo do objeto.
- → Void/Int colisão....: são funções que checam se um objeto está sobrepondo outro, ou seja, se houve colisão e previne que está aconteça, seja perdendo o jogo ou deletando o objeto.

Todas as funções:
int descobrirScore(char nome_arquivo[], int pontos_atual): essa função recebe como parâmetro o arquivo com o Record e a pontuação atual do jogador, checa se a pontuação atual é maior ou igual ao highscore. Se for maior a highscore passa a ser a atual. Atualiza arquivo de Record e retorna a highscore.
float distanciaPonto(float x1, float x2, float y1, float y2) : recebe dois pontos como parâmetro e retorna a distância entre eles.
void initEstrela(Estrela estrela[][NUM_ESTRELAS], int num_plano, int tam_estrelas) : recebe struct estrela e inicializa as estrelas(pixel) do plano de fundo.
void atualizaEstrela(Estrela estrela[][NUM_ESTRELAS], int num_plano, int tam_estrelas) : recebe struct estrela e dá movimento aos pixels(estrelas) do plano de fundo
void desenhaEstrela(Estrela estrela[][NUM_ESTRELAS], int num_plano, int tam_estrelas) : recebe struct estrela e desenha os pixels(estrelas)
void initNave(Nave *nave) : recebe struct nave e inicializa a nave
void atualizaNave(Nave *nave): recebe struct nave e dá movimento para a nave, depende de uma direção y e direção x que altera de acordo com as teclas que são pressionadas. O desenho da nave é feito no Main
void initBase(Base *base) : recebe struct base e inicializa a base(chão)
void atualizaBase(Base *base) : recebe struct base e atualiza a posição no cenário
void desenhaBase(Base base, ALLEGRO_BITMAP *chao) :recebe struct base e desenha a base
void initBloco(Bloco *bloco) : recebe struct bloco e inicializa os blocos
void atualizaBloco(Bloco *bloco) : recebe struct bloco e atualiza a posição desse bloco no cenário
void desenhaBloco(Bloco bloco, ALLEGRO_BITMAP *rocha1): recebe sctruct bloco e um bitmap, desenha o bloco com a imagem
void initTiro(Tiro tiro[], int tamanho): recebe struct tiro e inicializa o tiro
void atiraTiro(Tiro tiro[], int tamanho, Nave nave) : recebe struct tiro e struct nave, confirma se o tiro está ativo (caso o jogador tenha pressionado espaço) e posiciona o tiro na ponta da nave, além de colocar o tiro.ativo como true(1)
void atualizaTiro(Tiro tiro[], int tamanho): recebe struct tiro e atualiza a posição desse tiro no cenário, considerando que se sair da tela vai ser desativado(tiro.ativo=0)
void desenhaTiro(Tiro tiro[], int tamanho, ALLEGRO_BITMAP *tiro_sprite): recebe sctruct tiro e um bitmap, desenha o tiro com a imagem
void initTirao(Tirao *tirao) : recebe struct tirao(tiro mais forte) e inicializa o tirao
void atiraTiraoMelhor(Tirao *tirao, Nave nave) : recebe struct tirao e struct nave, confirma se o tirao está ativo (caso o jogador tenha pressionado e segurado o espaço) e posiciona o tirao na ponta da nave, além de colocar o tiro.ativo como true(1)

void atualizaTirao(Tirao *tirao, Nave nave, ALLEGRO_TIMER *timer): recebe struct tirao e struct nave e atualiza a posição desse tiro no cenário, considerando que se sair da tela vai ser desativado(tiro.ativo=0). E por ser o tiro mais forte, também tem como função fazer o tiro aumentar de tamanho até que esteja pronto para o ser atirado.
void desenhaTirao(Tirao tirao, ALLEGRO_BITMAP *tiro_sprite): recebe sctruct tirao e um bitmap, desenha o tirao, quando pronto, com a imagem e desenha ele como um círculo conforme for aumentando e ficando "pronto"
void initEnemy(Enemy enemy[], int tam): recebe struct enemy e inicializa o inimigo(cometa)
void soltaEnemy(Enemy enemy[], int tam, Nave *nave): recebe struct enemy e aleatoriamente seleciona se um enemy novo vai ser criado, qual o seu tamanho, qual sua velocidade e qual sua posição antes de entrar na tela. No final altera o enemy.ativo para 1(true) para indicar que o enemy foi ativado
void atualizaEnemy(Enemy enemy[], int tam) : recebe struct enemy atualiza a posição desse enemy no cenário, considerando que se sair da tela vai ser desativado(enemy.ativo=0).
void desenhaEnemy(Enemy enemy[], int tam, ALLEGRO_BITMAP *cometa): recebe sctruct enemy(cometa) e um bitmap, desenha o enemy com a imagem
int checarColisaoLado(Nave *nave, Base *base): recebe struct nave e struct base e checa se a nave vai se chocar com algum dos lados ou com a base. Caso se choque com a base (parte de baixo) a função vai retornar 1 para indicar que bateu (colisão true), nos outros casos vai bloquear a nave de sair para fora da tela e retornar 0.
int colisaoNaveBloco(Nave *nave, Bloco *bloco): recebe struct nave e struct bloco e confirma se houve colisão da nave com o bloco, caso aconteça a função retorna 1 que vai indicar que houve a colisão. O main ao receber isso redireciona o jogador ao menu final e acaba com o jogo.
int colisaoNaveEnemy(Nave *nave, Enemy enemy[], int tam) : recebe struct nave e struct enemy e checa se qualquer um dos inimigos colidiu com a nave. Caso aconteça a colisão a função retorna 1 que indica que houve a colisão. O main ao receber isso redireciona o jogador ao menu final e acaba com o jogo.
int colisaoTiroEnemy(Tiro tiro[], int tam_t, Enemy enemy[], int tam_e, Nave *nave): recebe struct tiro e struct enemy e checa se qualquer tiro se choca com qualquer inimigo. Caso aconteça a colisão a função retorna 1 que indica que houve a colisão. Tanto o enemy quanto o tiro são desativados e os pontos do jogador atualizados, indicando que ganhou pontos ao destruir cometa(enemy);
void colisaoTiroBloco(Tiro tiro[], int tam_t, Bloco bloco) : recebe struct tiro e struct bloco e checa se algum tiro se choca com algum bloco. Caso a colisão se confirme o tiro é desativado.
int colisaoTiraoEnemy(Tirao *tirao, Enemy enemy[], int tam_e, Nave *nave) : mesma coisa que o colisaoTiroEnemy, porém recebe o struct tirao
void colisaoTiraoBloco(Tirao *tirao, Bloco bloco) : mesma coisa que o colisaoTiroBloco, porém recebe o struct tirao
void colisaoEnemyBloco(Enemy enemy[], int tam_e, Bloco bloco) : mesma coisa que o colisaoTiroBloco, porém recebe o struct enemy
void colisaoEnemyEnemy(Enemy enemy[], int tam) : recebe struct enemy e checa se algum enemy se choca com algum outro enemy. Caso a colisão se confirme ambos enemys são desativados.

void initPowerUp(Power power[], int tam_p) : recebe struct power e inicializa os Power Ups
void soltarPowerUp(Power power[], int tam_p,ALLEGRO_TIMER *timer): recebe struct power e allegro timer. Essa função libera aleatoriamente, dependendo de um timer e de um rand(), os power up para o jogador, posicionando (também aleatoriamente) eles na tela e os alterando para ativos
void attPowerUp(Power power[], int tam_p, Bloco bloco, Nave *nave, ALLEGRO_BITMAP *powerShi, ALLEGRO_TIMER *timer): essa função recebe struct power, struct bloco, struct nave e um bitmap. Essa função atualiza a posição do power up na tela quando ativo, e também desenha ele a partir de uma imagem (bitmap recebido). Além disso ela checa se existe colisão da nave ou do bloco com o power, caso exista ela desativa o power up. Caso a colisão seja com a nave ativa a invencibilidade na nave por um certo periodo de tempo (controlado pelo timer).

- **TP_funcoes.h**

Cabeçalhos das funções.